10-2-00

A

JC915 U.S. PTO
09/29/00

Attorney's Docket: E20000370

JC899 U.S. PTO
09/676075
09/29/00

# PATENT APPLICATION TRANSMITTAL LETTER

**To the Assistant Commissioner for Patents:**

Transmitted herewith for filing is the patent application of: Gregory Bruce Gillooly, David Waters, Cheng-Yang Chou, Peter Lui, Samuel A. Mattoni, Jr., Edward S. Mallinak, Eric Robertson and Charles Stack

For: NETWORK COMMUNICATIONS COUPLER

**Enclosed are:**

- ☐ Combined Declaration and Power of Attorney
- ☒ one page cover sheet for patent application, 17 pages of specification, 8 pages of claim(s), and one page(s) of abstract
- ☒ five sheets of informal drawings
- ☐ Information Disclosure Statement
- ☐ Assignment, with Recordation Form Cover Sheet , of the invention to
- ☒ Postal card acknowledging receipt for filing

| CLAIMS AS FILED | | | Number Filed | Number Extra | | Rate | Fee |
|---|---|---|---|---|---|---|---|
| Total Claims | 18 | - | 20 = | ______ | x | $18.00 = | $________ |
| Independent Claims | 2 | - | 3 = | ______ | x | $78.00 = | $________ |
| Basic Fee | | | | | | | $690.00 |
| | | | | | | **Total Filing Fee** | $690.00 |

~ Certificate of Mailing Appears on the Following Page ~

**Method of Payment of Fees:**

☐ A check in the amount of $______ to cover the filing fee.

☒ Charge Deposit Account No. 05-0877 in the amount of $690.00 to cover the filing fee.. A duplicate copy of this Transmittal Letter is enclosed.

9/29/00
Date

Michael M. Rickin
Attorney of Record
Michael M. Rickin
Reg. No. 26,984
ABB Automation Inc.
29801 Euclid Avenue – 4U6
Wickliffe, OH 44092-1898
(440) 585-7840

I hereby certify that this patent application was deposited with the United States Postal Service EXPRESS MAIL NO.**TB842472873** in an envelope addressed to Box Patent Application, Commissioner for Patents, Washington, D.C. 20231 on the 29th day of September, **2000**.

Michael M. Rickin
Printed Name of Person Mailing Document

Michael M. Rickin
Signature of Person Mailing Document

Attorney's Docket: E20000370

# NETWORK COMMUINCATIONS COUPLER

Gregory Bruce Gillooly, David Waters, Cheng-Yang Chou, Peter Lui, Samuel A. Mattoni, Jr., Edward S. Mallinak, Eric Robertson and Charles Stack

I hereby certify that this patent application was deposited with the United States Postal Service EXPRESS MAIL NO. TB842472873 in an envelope addressed to BOX PATENT APPLICATION, Commissioner for Patents, Washington, D.C. 20231 on the 29th day of September, 2000.

Michael M. Rickin
Printed Name of Person Mailing Document

Michael M. Rickin
Signature of Person Mailing Document

# Network Communications Coupler

## 1. Field of the Invention

This invention relates to network communications couplers and more particularly to a coupler that performs its functionality in software.

## 2. Description of the Prior Art

A coupler is a communications device that maps data structures and operational behaviors between two dissimilar communication environments. In other words, a communications coupler interconnects a first network that uses a first protocol to a second network that uses a second protocol.

The basic structure of a typical communication coupler is well known to those of ordinary skill in the art and is shown in Fig. 1. Coupler 10 interconnects a first network 12 with a second network 14 and includes three software implemented functions in performing that interconnection. Two of these functions are a first communication protocol stack 16 and a second communication protocol stack 18. Stacks 16 and 18 provide the interface of the coupler 10 with the first and second networks 12 and 14, respectively.

The coupler 10 further includes a mapping program 20 that maps the protocols of the first and second networks 12 and 14 to each other. While not shown in Fig. 1, it is well known to those in the art that coupler 10 has to run on a suitable hardware platform and include suitably arranged circuit boards to connect the first protocol stack 16 to the first network 12 and the second protocol stack 18 to the second network 14.

One embodiment of a prior art communications coupler is the coupler sold by the assignee of the present invention that is used to interconnect the assignee's distributed process control systems (DCS) to the Ethernet. That coupler has a multiplicity of components including a personal computer that runs a server program, hardware to

interface with the communications network in the DCS, a network interface circuit, termination units, and a database engine for buffering for data and control information. This coupler while providing a very good interconnection between the DCS and the Ethernet does have a large footprint, consumes a fair amount of electrical power, requires multiple input power and cannot be adjacent to the process controlled by the DCS as the coupler is sensitive to environmental conditions.

In contrast thereto, the coupler of the present invention is an integrated solution that has a smaller footprint, uses less electrical power, has a single input power requirement, has a higher capacity and can be located adjacent to the process controlled by the DCS. Further the coupler of the present invention can be accessed externally using standard network protocol for maintenance purposes and allows for remote analysis of its performance.

<u>Summary of the Invention</u>

The present invention is a method for use with those of a plurality of commands sent by at least one client on a network connected to a network communications coupler designated by the at least one client as delayed response commands. The method has the steps of:

storing in the coupler a first instance of a template, the first instance used to store in the coupler at least one of the delayed response commands during execution of the at least one delayed response command by the coupler; and

storing in the coupler a second instance of the template, where the second instance is used to store in the coupler a reply to the at least one delayed response command executed by the coupler until the reply is retrieved by the client.

The present invention is also a method for use with those of a plurality of commands sent by at least one client

client on a network connected to a network communications coupler designated by the at least one client as delayed response commands. The method has the steps of:

designating by the at least one client which of the commands are delayed response commands;

storing in the coupler a first instance of a template, where the first instance is used to store in the coupler at least one of the delayed response commands during execution of the at least one delayed response command by the coupler; and

storing in the coupler a second instance of said template, where the second instance is used to store in the coupler a reply to the at least one delayed response command executed by the coupler until the reply is retrieved by the client.

## Description of the Drawing

Fig. 1 shows a typical implementation for a network communications coupler.

Fig. 2 shows a block diagram of the general architecture of the coupler of the present invention.

Fig. 3 shows a block diagram of one of the layers of Fig. 2.

Fig. 4 shows class FieldPoint and three examples of classes derived from that class.

Fig. 5 shows class ClientPoint and three examples of classes derived from that class.

Fig. 6 shows an instance of class ClientPoint that is associated with an instance of class FieldPoint.

Fig. 7 shows an example where various classes use the data of an associated access method of class FieldPoint.

## Description of the Preferred Embodiment(s)

Referring now to Fig. 2, there is shown a block diagram of the general architecture of a communications coupler 30 embodied in accordance with the present invention. Coupler 30 is used in this embodiment to interconnect a proprietary communications network 32 of a

DCS sold by the assignee of the present invention to a standard communication network 34 which in this one embodiment is the Ethernet network.

As is shown in Fig. 2, coupler 30 includes a mapping program 36 in the form of the server portion of a software program that functions to provide the protocol so that the coupler 30 can communicate with the proprietary hardware of the DCS in which coupler 30 is used. Coupler 30 further includes a first communication protocol stack 38 and a second communication protocol stack 40. First stack 38 includes first, second and third layers 38a, 38b and 38c respectively. Second layer 38b functions to interface with the operating system software 42 of coupler 30 and first layer 38a provides the interface between mapping program 36 and second layer 38b. Third layer 38c provides the drivers for the Ethernet.

Second stack 40 includes a first layer 40a and a second layer 40b. First layer 40a provides the interface between the mapping program 30 and the second layer 40b. Second layer 40b provides the control network interface between first layer 40a and the operating system software 42. A detailed description of layer 40b is given below in connection with Fig. 3.

Coupler 30 also includes a suitable hardware platform 44 which includes a digital processor and the circuits 46 and 48 to interface the coupler 30 with communication networks 32 and 34, respectively. In one embodiment for coupler 30, operating system software 42 is the VxWorks software available from Wind River Systems, Inc. of Alameda, CA.

Referring now to Fig. 3 there is shown a block diagram of the second layer 40b of second stack 40 of Fig. 2. Layer 40b communicates with standard communications network 34 based-clients through control network information server 50. Server 50 represents the combination of mapping program 36, first stacks 38, first

layer 40a and interface 48 of Fig. 2 and is an application that presents to the communications network 34 based clients a means by which control network 32 information may be exchanged between the clients and control network 32 resident entities. The control network information server 50 provides an information access service to the communications network 34 of Fig. 2.

The control network information server 50 communicates by commands and responses with a control network interface (CNI) command interpreter 52. The CNI Command Interpreter 52 provides the command/reply interface for data and command exchange between the software components of the control network interface (CNI) 76, which is essentially the second layer 40b of Fig. 2 and the client application. As is shown in Fig. 3 CNI 76 includes CNI command interpreter 52, control network device driver 72 and all of the blocks between interpreter 52 and driver 72.

The command/reply interface allows a client application to establish channels of communication to locations in the underlying control network. These locations may be individual data points or individual nodes or modules within the process control system. The CNI Command Interpreter 52 is responsible for recognizing the commands and requests, reformatting them as necessary and forwarding them to the appropriate subsystem component for processing. The CNI Command Interpreter 52 is also responsible for retrieving the requested information, formatting a proper reply to the client application and delivering it to the client on request.

The client may specify that it prefers a delayed (non blocking) response for certain commands sent by sender which commands are known as keyed commands. The keyed command management component 54 manages the execution of keyed commands and their replies. The processing of the keyed commands is begun at the request

of the client and when such processing is requested a keyed request is sent from the control network interface command interpreter 52 to the keyed command management component 54. A special reply which indicates that the command processing has started is supplied to the client by CNI Command Interpreter 52. During the command processing the client may proceed with other work. When the processing of the command is completed a reply to the client is generated that is stored in keyed reply storage 62, which manages the storage and retrieval of replies to keyed commands. The reply may be retrieved from by the client at the convenience of the client.

CNI command interpreter 52 also interfaces with node communications 56 which communicates through control network service provider 70, control network device driver 72 and control network hardware 74 with nodes in the remote control network to gather diagnostic or status information. Control network service provider 70 embodies the control network protocol. Control network device driver 72 provides the drivers that allows second layer 40b to communicate with the control network hardware 74. Control network hardware 74 implements the connection to the control network and thus provides the function of control network card 46.

As was described above, control network service provider 70 has the control network protocol and provides a convenient way to communicate with control network resident nodes.

Control network service provider 70 formats and sends control network messages, waits for replies and acknowledgements and supplies delivery confirmations, reply messages and message arrival notifications to the CNI 76. These reply messages are received by reply processing 58 which matches reply messages from the control network service provider 70 with outstanding requests from within the CNI 76. Reply processing 58

sends the received message to the requestor for further processing.

As was described above, control network device driver 72 provides the drivers that allows second layer 40b to communicate with the control network hardware 74 and thus offers a convenient way to communicate with the control network hardware. Control network device driver 72 initializes the control network hardware 74 and responds to interrupts from that hardware, buffers incoming and outgoing messages, and supplies a keep-alive handshake service to the control network hardware 74.

The remote control network includes various plug in circuit boards known as modules which are used in the control of the process. Module communications 60 communicates through control network service provider 70, control network device driver 72 and control network hardware 74 with those remote control network modules for the exchange of status, data and control information.

Nodes of the remote control network may change their status. For example, a node status may change to offline, online or busy. CNI 40b maintains records of the node status. These records are updated by node status monitor 64 which also notifies the export and import components 66 and 68, respectively, of the exception report (XR) database 67 of these changes.

The export XR database 66 holds current value, quality, status and timestamp for exception report points which are being sent from a Client (or the CNI itself) to the control network. Export XR database 66 maintains the connections between outgoing exception report points and control network-resident destinations. Import XR database 68 holds current value, quality, status and timestamp for exception report points which are being received from the control network for reporting to a Client. Import XR database 68 maintains the connections

between incoming exception report points and control network-resident sources.

A more detailed description of several of the blocks described above is given below. For the convenience of the reader each block or blocks is described in a section that carries an appropriate heading.

The XR Database 67

As was described above, the XR database 67 in the form of the Export XR and Import XR components 66, 68 stores data received from sources on one of the two networks 34, 36 and supplies it in a suitable form to requestors on the other of the two network 34, 36. Since the requestors may require the data in a different form than it is received from the source, the XR database 67 also provides the data translation methods.

To perform this function the XR database 67 represents each data source with an instance of the class named FieldPoint or a class derived from FieldPoint. A class derived from another class is also known as a subclass.

Fig. 4 shows the class FieldPoint 100 and three examples of classes 102, 104 and 106 derived from that class. The derived classes 102, 104 and 106 are named DigitalFieldPoint, AnalogFieldPoint, and HiResFieldPoint, respectively. The three derived classes 102, 104 and 106 are used to represent particular kinds of data sources. Each of the derived classes 102, 104 and 106 stores the most current value of the data from the network source 34or 36.

Three examples of data access methods are shown in the FieldPoint class 100: GetDigitalData() 100a, GetAnalogData() 100b, and GetHiResData() 100c, corresponding to the three types of data represented by the three derived classes 102, 104, 106. These data access methods 100a, 100b and 100c provide the data from the source in the specified form, translating it if

needed. The methods 100a, 100b and 100c are virtual methods, intended to be overridden in the derived classes 102, 104, 106.

Each subclass 102, 104, 106 provides (overrides) only the data access methods 100a, 100b, 100c appropriate to the data type it represents. Thus the DigitalFieldPoint class 102 has only the GetDigitalData() method 100a, while the AnalogFieldPoint and HiResFieldPoint classes 104, 106 have both the GetAnalogData() and GetHiResData() methods 100b, 100c, respectively.

Therefore it is possible to translate between the Analog and HiRes data formats, but not between the Digital format and either the Analog or HiRes formats. If a derived class 102, 104, 106 does not provide a particular method 100a, 100b and 100c, the base class FieldPoint 100 provides a version that may return an error status or other indication of a failed data access.

To get information from the other network, a client on one network creates an instance of a class named ClientPoint or a class derived from class ClientPoint. Fig. 5 shows class ClientPoint 110 and three examples of classes 112, 114 and 116 derived from that class. The derived classes 112, 114 and 116 are DigitalClientPoint, AnalogClientPoint, and HiResClientPoint, respectively. As in the FieldPoint classes, the three derived classes 112, 114 and 116 are used to represent particular kinds of data sources.

Class ClientPoint 112 and its three derived classes 112, 114 and 116 provide the access methods GetStatus() 110a and GetValue() 110b, for example, to retrieve information from or about the data source.

The instance of class ClientPoint 110 created by the client is associated with an instance of class FieldPoint 100 representing the desired data source, as shown in Fig. 6. The association is identified in Fig. 6 as

"portrays", as the instance of class ClientPoint 110 portrays the instance of class FieldPoint 100 for the client in a suitable form. Each ClientPoint is associated with one and only one FieldPoint, while each FieldPoint may have an association with zero, one, or more ClientPoints. Fig. 7 shows an example with one AnalogFieldPoint 104 associated with an AnalogClientPoint 114, a HiResClientPoint 116, and a DigitalClientPoint 112.

To retrieve information, a client uses, for example, the GetValue() methods 110b of the particular ClientPoint 110. The ClientPoint 110 then uses the particular FieldPoint 100 data access method 102, 104, 106 appropriate for its own data type to get the data from the FieldPoint 100.

In the example in Fig. 7, the AnalogClientPoint 114 uses the FieldPoint 100 data access method GetAnalogData() 100b, the HiResClientPoint 116 uses GetHiResData() 100c, and the DigitalClientPoint 112 uses GetDigitalData() 100a. If the FieldPoint 100 supports the data access method 100a, 100b, 100c used by the ClientPoint 112, 114, 116, i.e. it can provide the data in the correct form, it does so, otherwise it provides an error indication.

In the example shown in Fig. 7, the AnalogFieldPoint 104 is able to provide the data in the correct form to the AnalogClientPoint 114 and HiResClientPoint 116, but not to the DigitalClientPoint 112, as the AnalogFieldPoint 104 does not support the GetDigitalData() method 100a. Since the GetDigitalData() method 100a is not supported by the AnalogFieldPoint class 104, to provide the date to in correct form to the DigitalClientPoint 112, the base class FieldPoint 100 provides the default GetDigitalData() method 100a that supplies the error indication.

CNI 76

The CNI 76 is an object-oriented implementation. It presents to the control network information server 50 a command/reply interface equivalent to the command/reply interface of the prior art coupler sold by the assignee of the present invention, but offers it by way of a direct function call rather than some external interface thereby increasing efficiency and performance.

```
// interface class
class CNI:public ClientInterface
{
public:
   CNI();
   ~CNI();
   int processCommand( char * CommandText, int
CommandTextSize );
   int getReply( char * ReplyDestination, int
MaxReplySize );
};
```

The CNI class provides a means by which a Client Application may access the functionality in the CNI through a familiar interface. The interface uses a simple command/reply protocol, where a command is issued and then a reply is retrieved.

In the prior art coupler mentioned above this command was formatted in a buffer which then was sent through the "send" side of a physical interface such as a RS232 serial or SCSI port. The reply then arrives on the "receive" side of the physical interface when processing is completed by the device that received the sent command. The receive side has no control over when the device generates its reply.

This presentation has three benefits as follows:

a. The command is executed directly by invoking the processCommand method of a CNI object

The Client Application instantiates a CNI object,

constructs a command string and invokes the CNI processCommand method, passing in the command string. When the method returns, processing is complete. This is distinct from the prior art coupler mentioned above in that the Client Application in that coupler has to manage a hardware connection and tolerate all of its vagaries, as well as having no indication of the completion of command processing.

b. The reply is retrieved by the Client action of invoking the getReply method of the CNI object

The Client Application is informed of the completion of command execution by the returning of control from the CNI processCommand method. This also indicates a reply is ready for retrieval. The Client Application may at its own discretion retrieve the reply from the CNI that executed the command simply by invoking its getReply method, supplying as an argument a memory location to which the reply should be copied. This is distinct from the prior art coupler mentioned above in that the Client Application of that coupler has to manage a hardware connection and tolerate all of its vagaries, as well as having to accept the asynchronous arrival of a reply.

c. There may be more than one CNI object present simultaneously in the system

The CNI object implements an interface to the CNI 76. Unlike the prior art mentioned above, multiple CNI objects may be instantiated and function simultaneously.

The individual command handlers are implemented as objects derived from a common base class that encapsulates the command handling behaviors that are common to all commands, as well as support for asynchronous completion of command execution as required

by the keyed commands.

Keyed Command Management 54 and Keyed Reply Storage 62

As was described above, the client may specify that it prefers a delayed (non blocking) response for certain commands sent by the client which commands are known as keyed commands. The keyed command management component 54 manages the execution of keyed commands and their replies. When the processing of the keyed command is completed a reply to the client is generated that is stored in keyed reply storage 62, which manages the storage and retrieval of replies to keyed commands.

Keyed command management 54 includes a template, known as the keyed storage template which is shown below, for the storage and management of heap-resident objects where pointers to individual objects are stored and retrieved using a "key" or index value provided by the user, which in this embodiment is the client. The key or index value specified by the user is used for designating and retrieving a reply thereby allowing the user to match the rely with the keyed command.

Methods are provided for putting objects, getting objects and checking to see if an object is present under a certain "key". Storing of pointers eliminates copying. A user-supplied key provides an additional degree of control to the user (e.g. the user can reserve certain keys for particular uses).

The keyed storage template consists of a template declaration having one template parameter class (Entry) in the template parameter list and several statements defining the template.

Members comprise a pointer of type Entry, enumerated special values for designating boundary and special conditions, and several methods.

Methods comprise constructor, destructor and access methods. The constructor and destructor methods serve the usual purposes. One access method takes arguments of key

and Entry and has the purpose of placing the Entry into keyed storage associating it with the given key. One access method takes a single argument of a key and has the purpose of returning the corresponding Entry after performing a destructive read that removes the Entry from storage. One access method takes a single argument of a key and has the purpose of returning the corresponding entry without performing a destructive read, thus leaving the corresponding Entry in storage.

Specializations of this template are used in keyed command management 54 and keyed reply storage 62. The specialization of the keyed storage template used in keyed command management 54 stores delayed response commands while they are being executed, associating the command with the key value supplied by the client. When the command has completed execution, the specialization of the keyed storage template used in keyed reply storage 62 stores the corresponding reply to the delayed response command, associating the reply with the key value supplied by the client. The client specifies this key again when requesting retrieval of the reply.

```
//
// KeyedStorage Implementation
//
template <class Entry>
KeyedStorage<Entry>::KeyedStorage()
{
   storage = new Entry [1+MAXKEYVAL];

   {
      storage[i] = NULL;
   }
};

template <class Entry>
KeyedStorage<Entry>::~KeyedStorage()
{
   delete[] storage;
};

template <class Entry> int KeyedStorage<Entry>::putEntry(
int key, Entry newEntry )
{
   int success;
```

```
   if( key >=0 && key <= MAXKEYVAL )
   {
      storage[key] = newEntry;
      success = TRUE;
   }
   else
   {
      success = FALSE;
   }

   return( success );
};

template <class Entry> Entry
KeyedStorage<Entry>::getEntry( int key )
{
   Entry foundEntry;

   if( key >= 0 && key <= MAXKEYVAL )
   {
      foundEntry = storage[key];
      storage[key] = NULL;
   }
   else
   {
      foundEntry = NULL;
   }

   return( foundEntry );
};

template <class Entry> Entry
KeyedStorage<Entry>::checkEntry( int key )
{
   Entry foundEntry;

   if( key >= 0 && key <= MAXKEYVAL )
   {
      foundEntry = storage[key];
   }
   else
   {
      foundEntry = NULL;
   }

   return( foundEntry );
};

int KSTest(void )
{
   KeyedStorage<CNIReply *> ReplyStorage;
   KeyedStorage<CNICommand *> ActiveCommands;
   KeyedStorage<ClientInterface *> ClientInterfaces;
```

```
   CNIReply * Reply1 = (CNIReply *)1;
   CNIReply * Reply2 = (CNIReply *)2;

   CNICommand *Command1 = (CNICommand *)101;
   CNICommand *Command2 = (CNICommand *)102;

   ClientInterface *Client1 = (ClientInterface *)1001;
   ClientInterface *Client2 = (ClientInterface *)1002;

   ReplyStorage.putEntry( 1, Reply1 );
   ActiveCommands.putEntry( 1, Command1 );
   ClientInterfaces.putEntry( 1, Client1 );

   ReplyStorage.putEntry( 2, Reply2 );
   ActiveCommands.putEntry( 2, Command2 );
   ClientInterfaces.putEntry( 2, Client2 );

   printf("\n\ralles put\n\r");

   printf("\n\rchecking first reply = %p",
ReplyStorage.checkEntry( 1 ) );
   printf("\n\rchecking second reply = %p",
ReplyStorage.checkEntry( 2 ) );

   printf("\n\rchecking first command = %p",
ActiveCommands.checkEntry( 1 ) );
   printf("\n\rchecking second command = %p",
ActiveCommands.checkEntry( 2 ) );

   printf("\n\rchecking first client = %p",
ClientInterfaces.checkEntry( 1 ) );
   printf("\n\rchecking second client = %p",
ClientInterfaces.checkEntry( 2 ) );

   printf("\n\rgetting first reply = %p",
ReplyStorage.getEntry( 1 ) );
   printf("\n\rgetting first reply again = %p",
ReplyStorage.getEntry( 1 ) );

   printf("\n\rgetting second command = %p",
ActiveCommands.getEntry( 2 ) );
   printf("\n\rgetting second command again = %p",
ActiveCommands.getEntry( 2 ) );

   printf("\n\rgetting first client = %p",
ClientInterfaces.getEntry( 1 ) );
   printf("\n\rgetting first client again = %p",
ClientInterfaces.getEntry( 1 ) );

   printf("\n\r");

   return 0;
}
```

As of ordinary skill in the art can appreciate the keyed storage template is a software mechanism for the storage of pointers to a finite number of like software objects by association with a user-supplied key value, which mechanism has the following features:

- Maximizes speed of access by storing pointers rather than copying entire objects
- Minimizes system memory usage by storing pointers rather than entire objects
- putEntry( ) Stores pointer to object, associating it with user-supplied key value
- getEntry( ) Permits removal of an arbitrary pointer by specification of key value (destructive read)
- checkEntry( ) Permits retrieval of an arbitrary pointer by specification of key value (non-destructive read)

The template allows coupler 30 to support the storage and retrieval of replies to "keyed commands" and the management of command execution, including support of a "cancel" operation removing the reply from storage.

It is to be understood that the description of the preferred embodiment(s) is (are) intended to be only illustrative, rather than exhaustive, of the present invention. Those of ordinary skill will be able to make certain additions, deletions, and/or modifications to the embodiment(s) of the disclosed subject matter without departing from the spirit of the invention or its scope, as defined by the appended claims.

What is claimed is:

1. A method for use with those of a plurality of commands sent by at least one client on a network connected to a network communications coupler designated by said at least one client as delayed response commands comprising the steps of:

storing in said coupler a first instance of a template, said first instance used to store in said coupler at least one of said delayed response commands during execution of said at least one delayed response command by said coupler; and

storing in said coupler a second instance of said template, said second instance used to store in said coupler a reply to said at least one delayed response command executed by said coupler until said reply is retrieved by said client.

2. The method of Claim 1 wherein said template has a declaration having one template parameter class in the parameter list of said template and one or more statements defining said template.

3. The method of Claim 2 wherein said one or more statements defining said template include a pointer of said template parameter class.

4. The method of Claim 3 wherein said one or more statements defining said template further include enumerated special values for designating boundary and special conditions and one or more methods.

5. The method of Claim 4 wherein said one or more methods statements comprise constructor, destructor and access methods.

6. The method of Claim 2 wherein said template is:

```
template <class Entry>
KeyedStorage<Entry>::KeyedStorage()
{
   storage = new Entry [1+MAXKEYVAL];

   {
      storage[i] = NULL;
   }
```

```
};

template <class Entry>
KeyedStorage<Entry>::~KeyedStorage()
{
   delete[] storage;
};

template <class Entry> int KeyedStorage<Entry>::putEntry(
int key, Entry newEntry )
{
   int success;

   if( key >=0 && key <= MAXKEYVAL )
   {
      storage[key] = newEntry;
      success = TRUE;
   }
   else
   {
      success = FALSE;
   }

   return( success );
};

template <class Entry> Entry
KeyedStorage<Entry>::getEntry( int key )
{
   Entry foundEntry;

   if( key >= 0 && key <= MAXKEYVAL )
   {
      foundEntry = storage[key];
      storage[key] = NULL;
   }
   else
   {
      foundEntry = NULL;
   }

   return( foundEntry );
};

template <class Entry> Entry
KeyedStorage<Entry>::checkEntry( int key )
{
   Entry foundEntry;

   if( key >= 0 && key <= MAXKEYVAL )
   {
      foundEntry = storage[key];
   }
   else
```

```
    {
       foundEntry = NULL;
    }

    return( foundEntry );
};

int KSTest(void )
{
    KeyedStorage<CNIReply *> ReplyStorage;
    KeyedStorage<CNICommand *> ActiveCommands;
    KeyedStorage<ClientInterface *> ClientInterfaces;

    CNIReply * Reply1 = (CNIReply *)1;
    CNIReply * Reply2 = (CNIReply *)2;

    CNICommand *Command1 = (CNICommand *)101;
    CNICommand *Command2 = (CNICommand *)102;

    ClientInterface *Client1 = (ClientInterface *)1001;
    ClientInterface *Client2 = (ClientInterface *)1002;

    ReplyStorage.putEntry( 1, Reply1 );
    ActiveCommands.putEntry( 1, Command1 );
    ClientInterfaces.putEntry( 1, Client1 );

    ReplyStorage.putEntry( 2, Reply2 );
    ActiveCommands.putEntry( 2, Command2 );
    ClientInterfaces.putEntry( 2, Client2 );

    printf("\n\ralles put\n\r");

    printf("\n\rchecking first reply = %p",
ReplyStorage.checkEntry( 1 ) );
    printf("\n\rchecking second reply = %p",
ReplyStorage.checkEntry( 2 ) );

    printf("\n\rchecking first command = %p",
ActiveCommands.checkEntry( 1 ) );
    printf("\n\rchecking second command = %p",
ActiveCommands.checkEntry( 2 ) );

    printf("\n\rchecking first client = %p",
ClientInterfaces.checkEntry( 1 ) );
    printf("\n\rchecking second client = %p",
ClientInterfaces.checkEntry( 2 ) );

    printf("\n\rgetting first reply = %p",
ReplyStorage.getEntry( 1 ) );
    printf("\n\rgetting first reply again = %p",
ReplyStorage.getEntry( 1 ) );

    printf("\n\rgetting second command = %p",
ActiveCommands.getEntry( 2 ) );
```

```
   printf("\n\rgetting second command again = %p",
ActiveCommands.getEntry( 2 ) );

   printf("\n\rgetting first client = %p",
ClientInterfaces.getEntry( 1 ) );
   printf("\n\rgetting first client again = %p",
ClientInterfaces.getEntry( 1 ) );

   printf("\n\r");

   return 0;
}
```

7. The method of Claim 1 further comprising the step of defining for said template a declaration having one template parameter class in the parameter list of said template and one or more statements defining said template.

8. The method of Claim 7 further comprising the step of defining for said one or statements a pointer of said one template parameter class, enumerated special values for designating boundary and special conditions, and one or more methods.

9. The method of 8 further comprising the step of defining for said one or statements that are method statements methods selected from constructor, destructor and access methods.

10. A method for use with commands sent to a network communications coupler by at least one client on a network connected to said coupler comprising the steps of:

designating by said at least one client which of said commands are delayed response commands,

storing in said coupler a first instance of a template, said first instance used to store in said coupler at least one of said delayed response commands during execution of said at least one delayed response command by said coupler; and

storing in said coupler a second instance of said template, said second instance used to store in said coupler a reply to said at least one delayed response

command executed by said coupler until said reply is retrieved by said client.

11. The method of Claim 10 wherein said template has a declaration having one template parameter class in the parameter list of said template and one or more statements defining said template.

12. The method of Claim 11 wherein said one or more statements defining said template include a pointer of said template parameter class.

13. The method of Claim 12 wherein said one or more statements defining said template further include enumerated special values for designating boundary and special conditions and one or more methods.

14. The method of Claim 13 wherein said one or more methods statements comprise constructor, destructor and access methods.

15. The method of Claim 11 wherein said template is:

```
template <class Entry>
KeyedStorage<Entry>::KeyedStorage()
{
   storage = new Entry [1+MAXKEYVAL];

   {
      storage[i] = NULL;
   }
};

template <class Entry>
KeyedStorage<Entry>::~KeyedStorage()
{
   delete[] storage;
};

template <class Entry> int KeyedStorage<Entry>::putEntry(
int key, Entry newEntry )
{
   int success;

   if( key >=0 && key <= MAXKEYVAL )
   {
      storage[key] = newEntry;
      success = TRUE;
   }
   else
```

```
   {
      success = FALSE;
   }

   return( success );
};

template <class Entry> Entry
KeyedStorage<Entry>::getEntry( int key )
{
   Entry foundEntry;

   if( key >= 0 && key <= MAXKEYVAL )
   {
      foundEntry = storage[key];
      storage[key] = NULL;
   }
   else
   {
      foundEntry = NULL;
   }

   return( foundEntry );
};

template <class Entry> Entry
KeyedStorage<Entry>::checkEntry( int key )
{
   Entry foundEntry;

   if( key >= 0 && key <= MAXKEYVAL )
   {
      foundEntry = storage[key];
   }
   else
   {
      foundEntry = NULL;
   }

   return( foundEntry );
};

int KSTest(void )
{
   KeyedStorage<CNIReply *> ReplyStorage;
   KeyedStorage<CNICommand *> ActiveCommands;
   KeyedStorage<ClientInterface *> ClientInterfaces;

   CNIReply * Reply1 = (CNIReply *)1;
   CNIReply * Reply2 = (CNIReply *)2;

   CNICommand *Command1 = (CNICommand *)101;
   CNICommand *Command2 = (CNICommand *)102;
```

```
   ClientInterface *Client1 = (ClientInterface *)1001;
   ClientInterface *Client2 = (ClientInterface *)1002;

   ReplyStorage.putEntry( 1, Reply1 );
   ActiveCommands.putEntry( 1, Command1 );
   ClientInterfaces.putEntry( 1, Client1 );

   ReplyStorage.putEntry( 2, Reply2 );
   ActiveCommands.putEntry( 2, Command2 );
   ClientInterfaces.putEntry( 2, Client2 );

   printf("\n\ralles put\n\r");

   printf("\n\rchecking first reply = %p",
ReplyStorage.checkEntry( 1 ) );
   printf("\n\rchecking second reply = %p",
ReplyStorage.checkEntry( 2 ) );

   printf("\n\rchecking first command = %p",
ActiveCommands.checkEntry( 1 ) );
   printf("\n\rchecking second command = %p",
ActiveCommands.checkEntry( 2 ) );

   printf("\n\rchecking first client = %p",
ClientInterfaces.checkEntry( 1 ) );
   printf("\n\rchecking second client = %p",
ClientInterfaces.checkEntry( 2 ) );

   printf("\n\rgetting first reply = %p",
ReplyStorage.getEntry( 1 ) );
   printf("\n\rgetting first reply again = %p",
ReplyStorage.getEntry( 1 ) );

   printf("\n\rgetting second command = %p",
ActiveCommands.getEntry( 2 ) );
   printf("\n\rgetting second command again = %p",
ActiveCommands.getEntry( 2 ) );

   printf("\n\rgetting first client = %p",
ClientInterfaces.getEntry( 1 ) );
   printf("\n\rgetting first client again = %p",
ClientInterfaces.getEntry( 1 ) );

   printf("\n\r");

   return 0;
}
```

16. The method of Claim 10 further comprising the step of defining for said template a declaration having one template parameter class in the parameter list of said template and one or more statements defining said template.

17. The method of Claim 16 further comprising the step of defining for said one or statements a pointer of said one template parameter class, enumerated special values for designating boundary and special conditions, and one or more methods.

18. The method of 17 further comprising the step of defining for said one or statements that are method statements methods selected from constructor, destructor and access methods.

## ABSTRACT

A coupler for connecting a first communications network to a second communications network. The coupler implements its functionality in software.

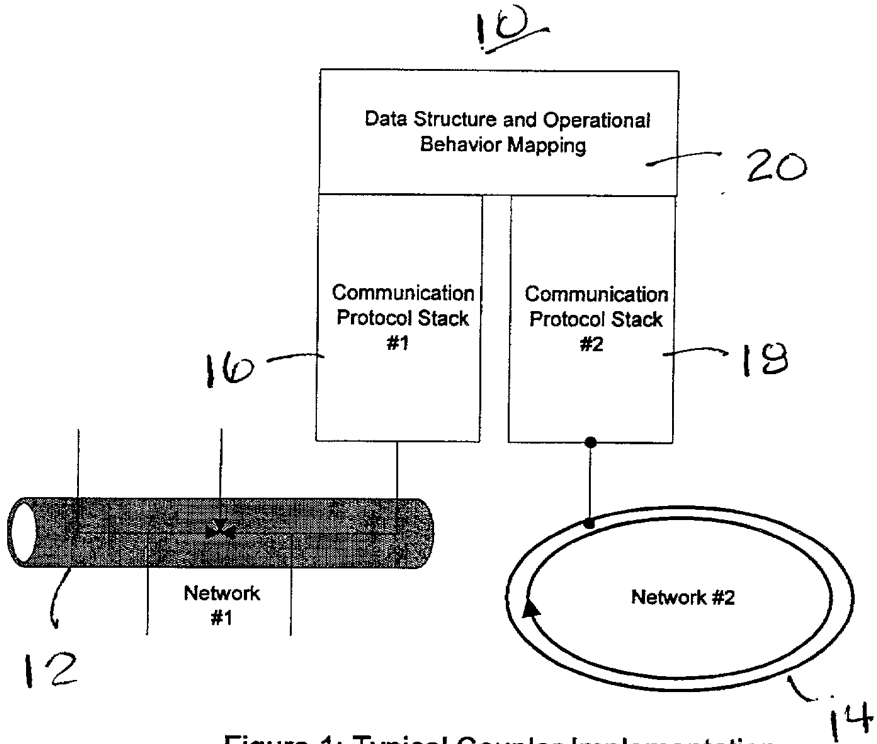
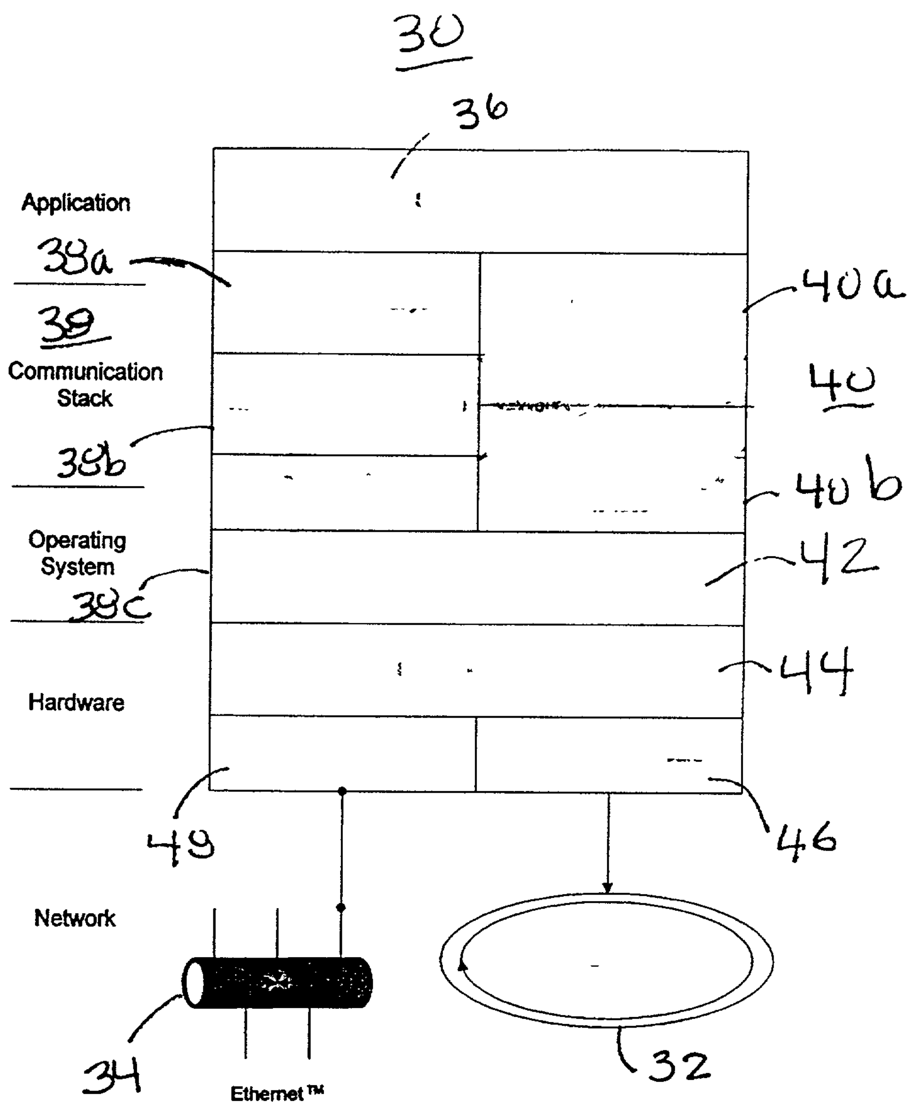



Figure 1: Typical Coupler Implementation

**Figure 2**


30
36
Application
38a
38
Communication Stack
38b
Operating System
38c
Hardware
Network
34
Ethernet™
40a
40
40b
42
44
46
49
32

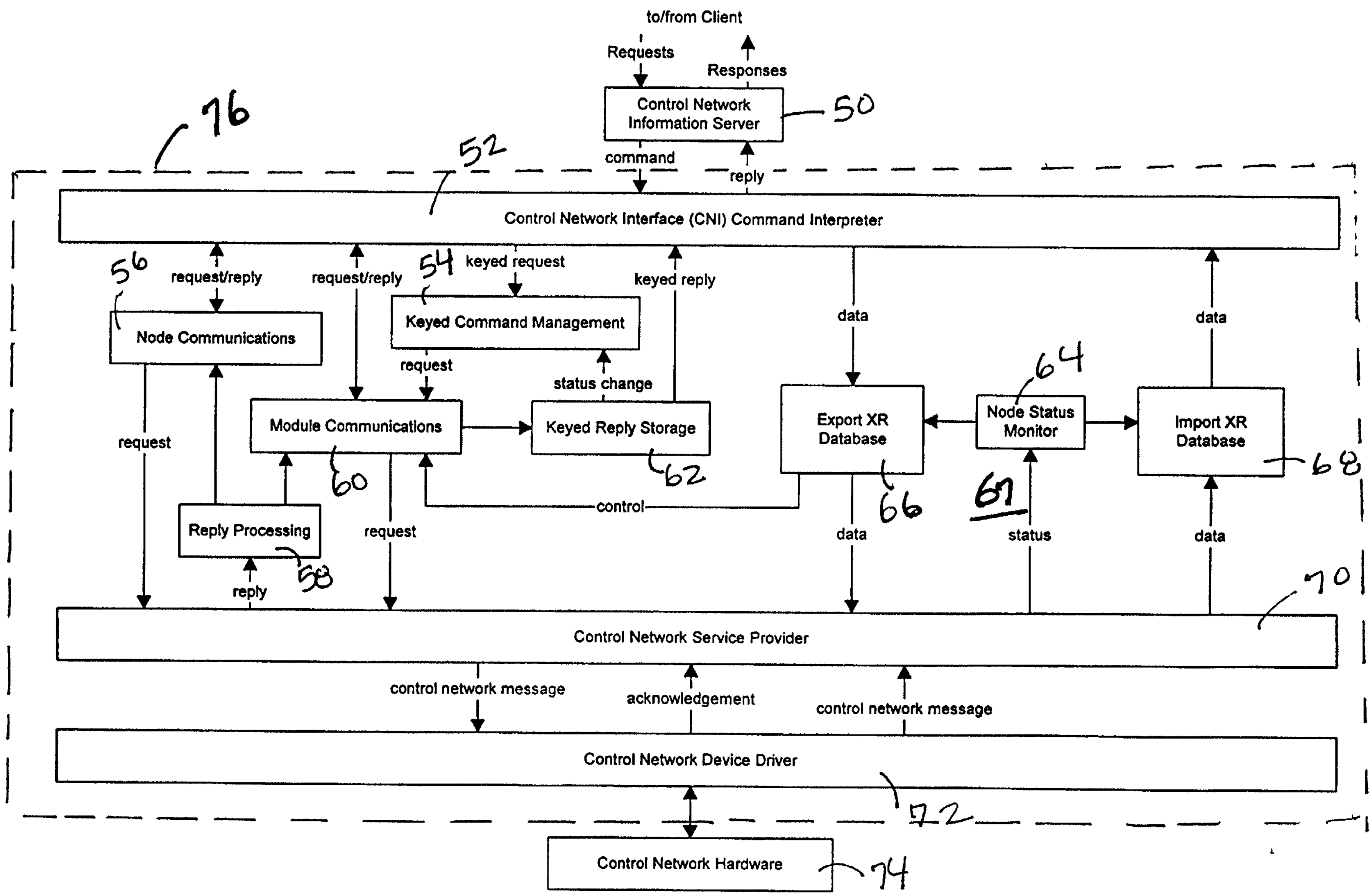

to/from Client
Requests
Responses
Control Network Information Server
50
76
52
command
reply
Control Network Interface (CNI) Command Interpreter
56
request/reply
request/reply
54
keyed request
keyed reply
Node Communications
Keyed Command Management
request
status change
data
data
64
Module Communications
Keyed Reply Storage
Export XR Database
Node Status Monitor
Import XR Database
68
request
60
62
control
66
61
Reply Processing
request
data
status
data
58
reply
70
Control Network Service Provider
control network message
acknowledgement
control network message
Control Network Device Driver
72
Control Network Hardware
74

FIG. 3

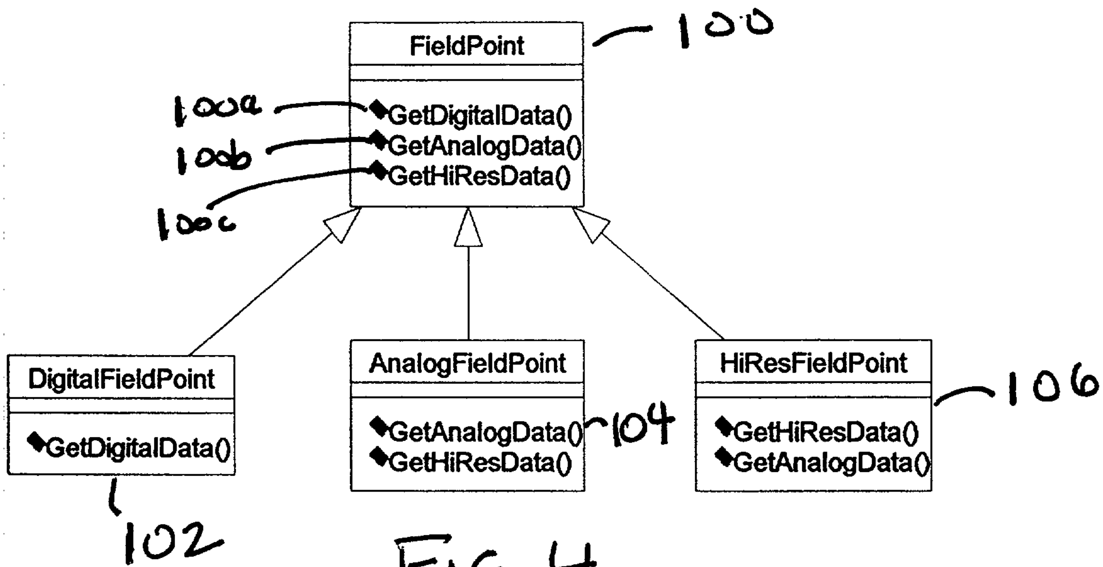

FieldPoint
GetDigitalData()
GetAnalogData()
GetHiResData()
100
100a
100b
100c
DigitalFieldPoint
GetDigitalData()
102
AnalogFieldPoint
GetAnalogData()
GetHiResData()
104
HiResFieldPoint
GetHiResData()
GetAnalogData()
106

FIG. 4

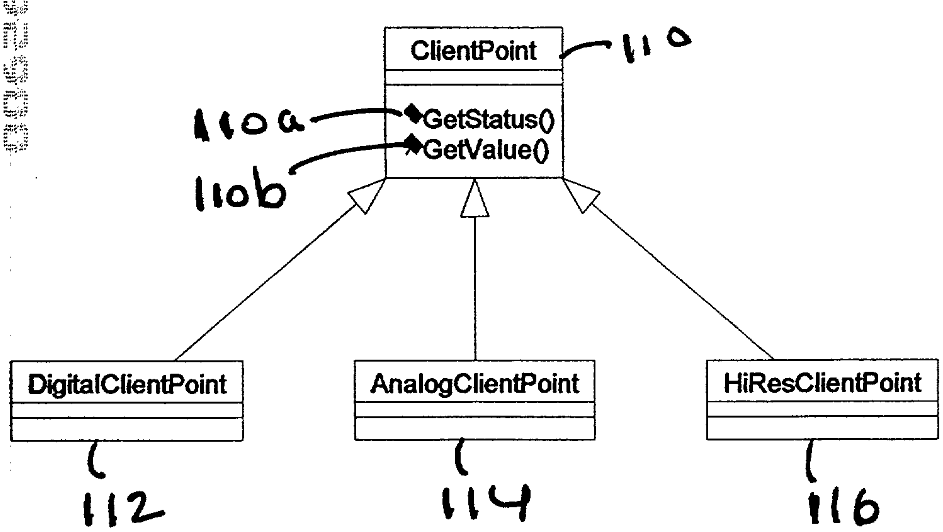

ClientPoint
GetStatus()
GetValue()
110
110a
110b
DigitalClientPoint
112
AnalogClientPoint
114
HiResClientPoint
116

FIG. 5

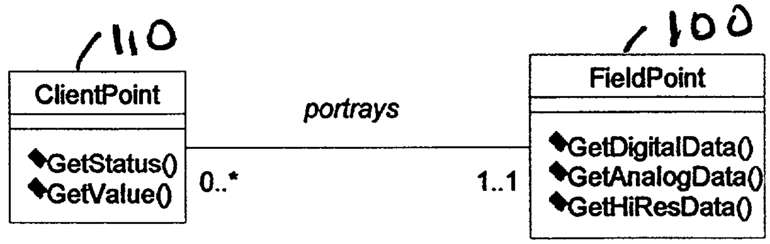

110
ClientPoint
GetStatus()
GetValue()
portrays
0..*
1..1
100
FieldPoint
GetDigitalData()
GetAnalogData()
GetHiResData()

FIG. 6

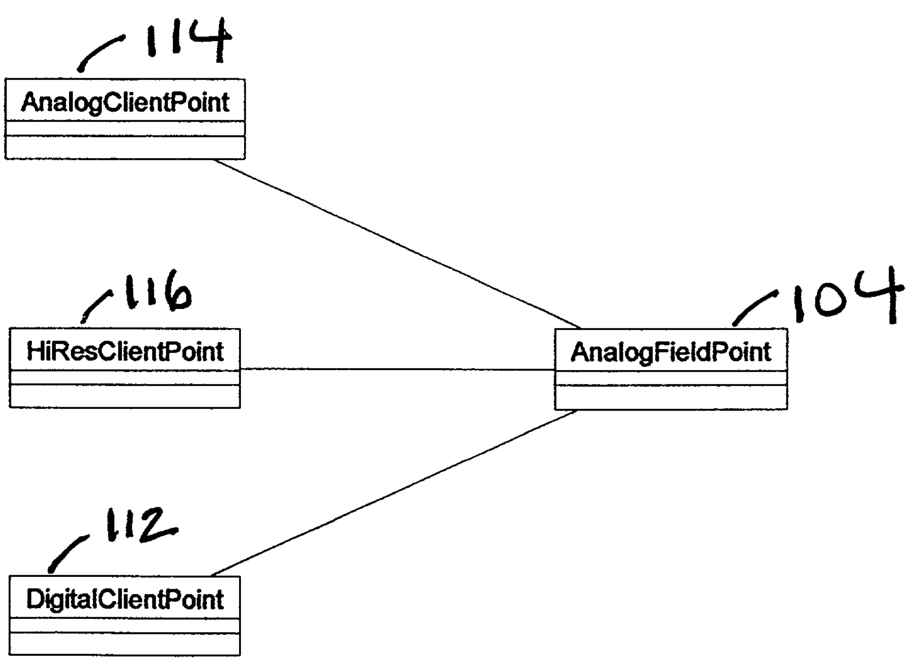

114
AnalogClientPoint
116
HiResClientPoint
112
DigitalClientPoint
104
AnalogFieldPoint

FIG. 7